

# Параллельное программирование. 1 семестр, задание 2.

## Отчет по выполнению работы.

### Описание программы

Структура программы

Программа выполнена с использованием объектно-ориентированного подхода.

Основные классы:

**ConfigReader** — читает параметры из конфигурационного файла.

**FileGenerator** — генерирует файл с натуральными числами от 1 до  $N$ .

**Utils** — вспомогательный класс для считывания данных из файла.

**SequentialProcessor** — выполняет последовательную обработку массива.

**ParallelProcessor** — выполняет многопоточную обработку массива:

- Равномерное распределение данных по потокам.
- Неравномерное распределение данных.

**ComplexOperations** — реализует сложные операции, такие как:

- Вычисление факториала.
- Вычисление чисел Фибоначчи.
- Возведение в степень.

Логика работы.

Генерация данных: Создается файл с натуральными числами от 1 до  $N$ .

Считывание данных: Файл загружается в массив.

Обработка данных:

Последовательная обработка (простая и сложная задачи).

Многопоточная обработка (равномерная и неравномерная).

Замер времени:

Для каждой конфигурации фиксируется время выполнения.

### Результаты исследования.

1. Почему эффект от распараллеливания наблюдается только при большем числе элементов?

Распараллеливание накладывает дополнительные издержки на создание и управление потоками. Для маленьких массивов время на синхронизацию потоков

превышает выигрыш от параллельной обработки. На рисунках 1 и 2 представлены графики, отображающие время работы (ось Y) относительно количества чисел (ось X) в исходном файле. Здесь было взято для распараллеливания 4 потока. Это является наглядным подтверждением ответа на поставленный вопрос. При выполнении сложной задачи мы видим, что после 100000 чисел время выполнения в последовательном и многопоточном режимах начинает сильно расходиться.

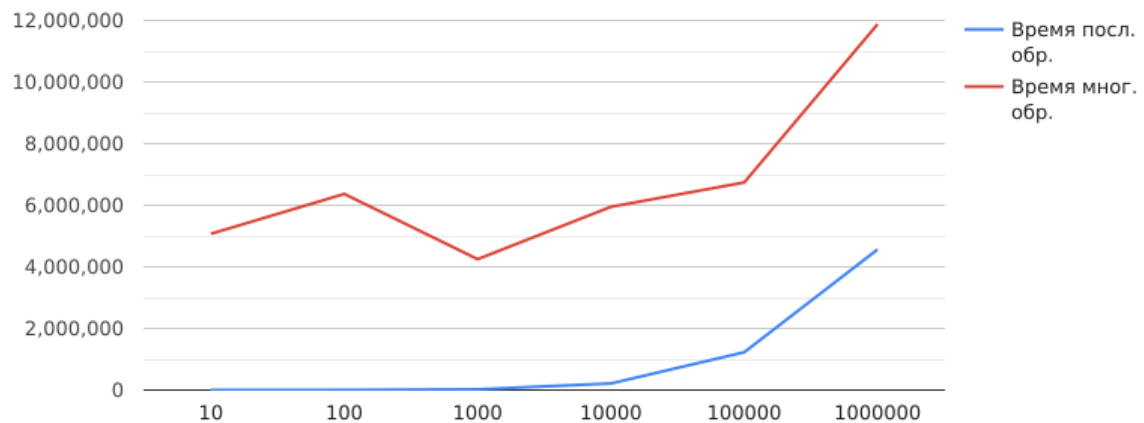


Рис. 1 - Простая задача

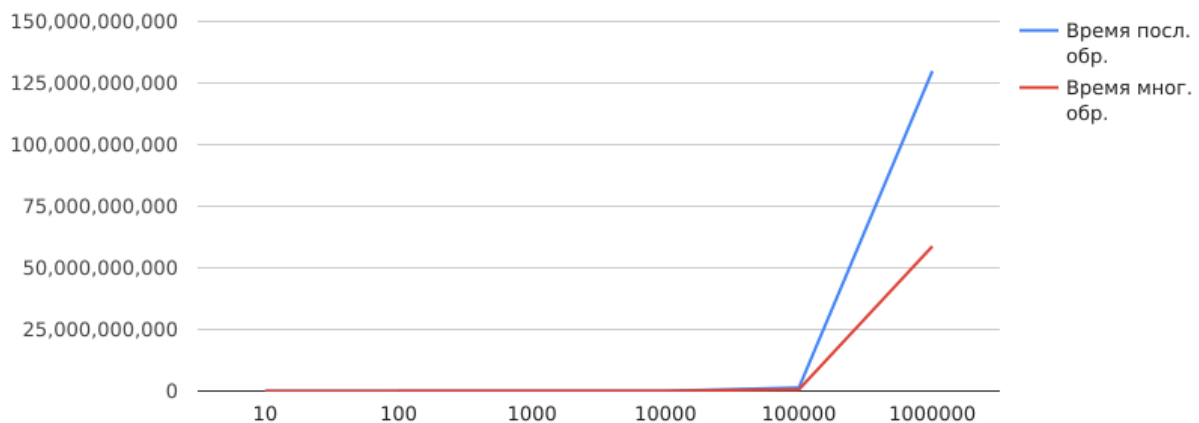


Рис. 2 - Сложная задача

## 2. Как влияет увеличение сложности обработки на эффективность многопоточной обработки?

Увеличение сложности задач (например, вычисление факториала) повышает выигрыш от распараллеливания, так как относительная стоимость управления потоками становится незначительной по сравнению с временем выполнения сложных операций.

В этом можно убедиться взглянув на рисунки 1 и 2. Мы получили выигрыш в производительности от многопоточной обработки данных только при выполнении сложной задачи. Такого не произошло при выполнении простой задачи (умножение на число) потому что время на синхронизацию потоков превышает выигрыш от параллельной обработки.

3. Какое число потоков является оптимальным для конкретной вычислительной системы? Как его подобрать?

Оптимальное количество потоков равно количеству физических или логических ядер процессора. Его можно подобрать эмпирически, варьируя  $M$  и измеряя время выполнения. На рисунке 3 представлено наглядно увеличение ускорения производительности (ось  $Y$ ) относительно числа потоков (ось  $X$ ).

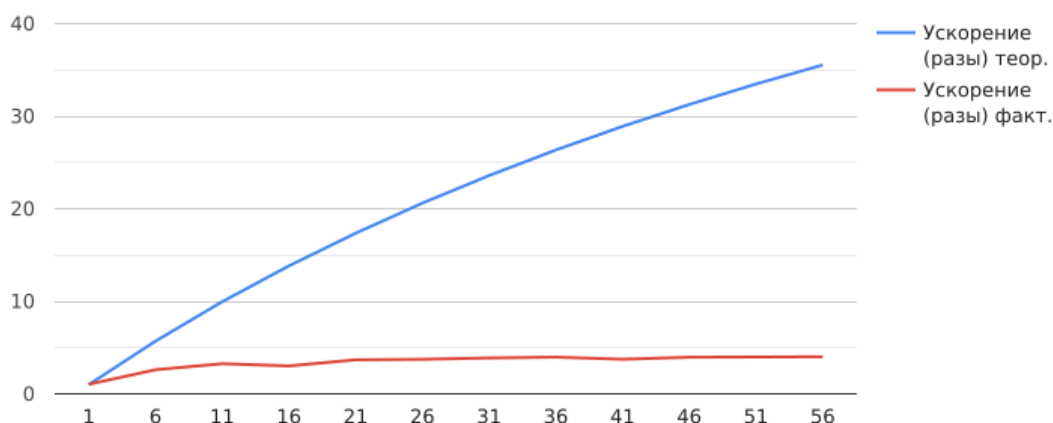


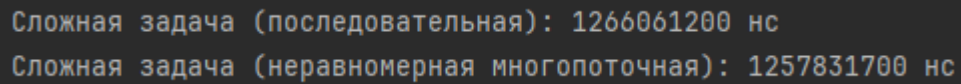
Рис.3 - Теоритическое и фактическое ускорение относительно количества потоков.

Синий график это теоретическое ускорение согласно закону Амдала, а красный фактическое ускорение. Мы видим что после определенного количества потоков красный график стабилизируется на примерно одном ускорении и производительность перестает расти. Эта точка и есть фактическое самое оптимальное количество потоков.

4. Почему неравномерность загрузки потоков приводит к снижению эффективности многопоточной обработки?

При неравномерном распределении данных один поток может завершить работу быстрее других, в то время как остальные продолжают выполнение, что приводит к недоиспользованию ресурсов процессора.

Это можно увидеть при сравнении времени выполнения программы для последовательной обработки сложной задачи и многопоточной обработки (количество потоков 4, распределение по потокам: 10, 10, 10, 99970 элементов).



```
Сложная задача (последовательная): 1266061200 нс  
Сложная задача (неравномерная многопоточная): 1257831700 нс
```

Рис 4. - Неравномерная многопоточная обработка

Фактически многопоточная обработка отработала как последовательная, потому что первые три потока с меньшим количеством элементов закончили свое выполнение и ждали окончания обработки последним потоком своих элементов. Таким образом три потока просто нерационально простаивали.

5. Как логичнее всего реализовать обработку таких данных?

Использовать равномерное распределение данных по потокам, учитывая сложность обработки. Для неравномерных задач рекомендуется динамическое распределение данных с использованием пула потоков.